A Secure Multi-Application Platform for Vehicle Telematics

Jef Maerien, Sam Michiels, Stefan Van Baelen, Christophe Huygens, Wouter Joosen
IBBT-DistriNet
Department of Computer Science
Katholieke Universiteit Leuven
B-3001, Leuven, Belgium
Email: firstname.lastname@cs.kuleuven.be

***Abstract*—Contemporary vehicles offer an advanced telematics platform with multiple applications available such as electronic road tolling, emergency call, breakdown call, or route planning. Given the sensitivity of collected driver and vehicle data and the potential use of this data by third party applications, security mechanisms are needed to protect services as well as data. Although security has been investigated in recent telematics studies, they do not consider potentially malicious or erroneous third party applications. This paper presents a secure multi-application platform that is designed as a modular software architecture with security features to support availability, confidentiality and integrity; a proof-of-concept prototype was developed on state-of-the-art hardware and software.**

I. Introduction

Intelligent vehicles offer a on-board telematics platform that not only controls functional features (e.g. automated break, or slip control) [1], but also provides a platform for applications to regulate and protect drivers (e.g. electronic road tolling (e-toll), emergency call (e-call), breakdown call) and a wide range of infotainment applications (e.g. traffic information service, video/radio streaming service, or route planner) [2].

Current research typically focuses on single applications. In reality however, all proposed applications co-operate in a limited and shared environment on a common telematics platform, and multiple independent parties are interested to use the data collected by such a platform. Although some research has investigated multi-application platforms, it focuses on applications with uniform requirements without addressing intrinsic security issues.

In this paper, we use the case of e-toll to illustrate that (1) the functional requirement of independent access to vehicle data is strong, and (2) non-functional requirements, such as security or privacy, are equally important. The concept of e-toll implies that road charging is based on the time and route driven instead of a flat fee based on car parameters or total mileage. In order to achieve this, the route is electronically monitored by a GPS tracker; journey information is then locally processed into an amount of toll due, or directly transmitted to the proper authorities to perform remote processing.

Other applications are equally interested in using the collected e-toll data to create new services. For example, a pay-as-you-drive insurance application that calculates insurance fees based on time and type of roads used, or an on-board payment application that allows the driver or a passenger to pay toll fees immediately after a trip.

The e-toll application also illustrates the need for three security measures: it must be running at all times (availability), it must collect and process sensitive journey data (confidentiality), and this data must be protected from tampering or unauthorized access (integrity) by other applications or parties, including the user.

The contribution of this paper is the definition and realisation of software architecture for a multi-application vehicle platform that addresses the three security requirements (availability, confidentiality, integrity). The architecture defines three features: (1) the separation between a Core Runtime Environment (CRE) and a Service Runtime Environment (SRE), (2) a Local Entity Access Control (LEAC) for the SRE, and (3) a Remote Entity Access Control (REAC) for the CRE and SRE.

The remainder of this paper is structured as follows: Section II positions the architecture of this paper in the wider GST architecture. Section III presents the three security requirements and relates them to the most relevant involved parties. Section IV presents the architecture of our secure multi-application platform. Section V evaluates a proof of concept prototype of the architecture. Section VI positions our work in the field of telematics software platforms and compares our solution to OSGi and the GST architecture. Section VII summarizes the main contributions in this paper and highlights future work.

II. Reference architecture

We position the proposed multi-application platform in the GST reference architecture for telematics [3] (see Figure 1). This architecture enables remote management of vehicle applications by a trusted party, the Host Management Center (HMC). We assume that vehicle manufacturers provide users with a platform to install and update applications. Additionally this would allow them to remotely manage functional software. Field upgradability is an increasingly important feature. In December 2009, Toyota had to recall several types of cars due to faulty brake software. Toyota estimated a loss of 2 billion dollars due to costs and lost sales from the recall [4].

Service Centers provide applications to the HMC; the HMC is responsible for verification of the software and deployment on to the vehicle platform. Due to the potential amount of

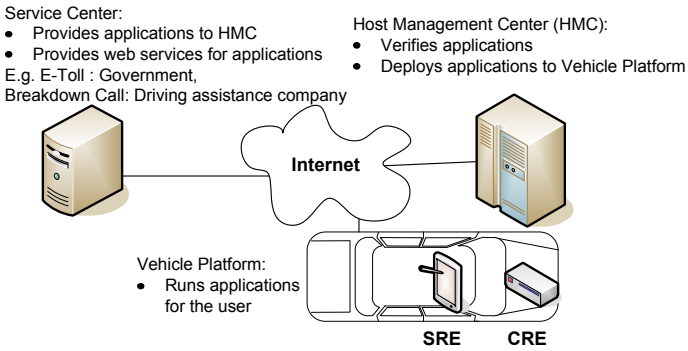

Fig. 1. Distributed reference architecture of the multi-application platform.

programs, this verification is automated. It is very unlikely that this static filter detects all erroneous or malicious programs. Some malicious or erroneous programs could be installed onto vehicle platforms.

Vehicle applications may need additional services from the Internet. For example, a traffic information application has a local component on the vehicle that informs the driver that updated traffic conditions can be downloaded; these additional services are also provided by a Service Center.

III. Security requirements

We identify three security requirements (availability, confidentiality and integrity) to be supported by the vehicle telematics platform and describe their role for different stakeholders involved: users, the government as third party service provider, and the HMC as management center. We apply the three security requirements on the applications installed on the platform, and two of them (confidentiality and integrity) on the data present on the platform.

- **Availability of applications**: Mandatory applications (e.g. e-toll, crash prevention) must have guarantees on the availability and timeliness of their operation, which will be required by law; infotainment applications (e.g. traffic information) may only require a best effort availability. Users however depend on the availability of certain infotainment services for guiding them and experience any downtime as a significant inconvenience.
- **Confidentiality of applications**: No application should be able to see which other applications are running on the application platform. An application should only be able to access and view those services it requires.
- **Integrity of applications**: Applications need to be protected from illegal modifications. It must be prevented that any party, including the user, can illegally alter any application. Government required applications will require by law to be protected from such modifications. The HMC is responsible for the component composition on the platform and must enforce correct and incorruptible operational conditions at all times.
- **Confidentiality of data**: Data must be protected from unauthorized access by third party applications. Users do not want that sensitive data on the vehicle can be read by non-authorized parties. The government requires that privacy sensitive information is adequately protected from malicious parties.
- **Integrity of data**: Data must be protected from illegal modifications. No party or application, including the user, should be able to modify any data it is not allowed to modify. For example users should not be able to change the toll they are due.

IV. Architecture

This section presents the software architecture of the secure application platform. We assume multiple applications are running on the platform. Each applications consists of components that implement the functional behaviour of the platform. Applications can communicate with each other through message passing. Internet/VANET access and data storage capabilities are available on the platform.

In order to meet the security requirements, we propose an architecture with three key features (see Figure 2): (1) the separation of the platform in two runtime environments, (2) a Local Entity Access Control (LEAC), and (3) a Remote Entity Access Control (REAC). The LEAC and REAC are policy driven. These security policies dictate to which resources respectively the applications or remote communication entities have access.

We assume that the platform is managed by the HMC. The HMC can manage the application platform in two ways. (1) The HMC can send new or updated applications to the application platform to provide new or improved functional behaviour. (2) The HMC can send new or updated security policies to the application platform to change the security enforcement of the platform. The HMC can manage the applications and policies by sending the new data to the Application Management or Policy Management application on the application platform.

A. Separation of runtimes

The first feature of the architecture is the separation of the multi-application platform in two separated platforms with independent hardware and runtime environments. The Core Runtime Environment (CRE) runs security critical or mandatory applications; the Service Runtime Environment (SRE) runs optional and non-critical applications.

The separation of runtimes supports *the availability, confidentiality and integrity of applications and the integrity and confidentiality of data*. Critical or mandatory applications and data on the CRE cannot be illegally affected by the SRE. It also enables the SRE to be more open without affecting the applications of the CRE.

Each platform has the availability to communicate over the Internet and over the local VANET through a remote communication service, and the SRE and CRE can communicate with each other through a local communication service. All communication over these channels must be encrypted to preserve *the integrity and confidentiality of the data*. As we illustrated, many infotainment applications are interested in

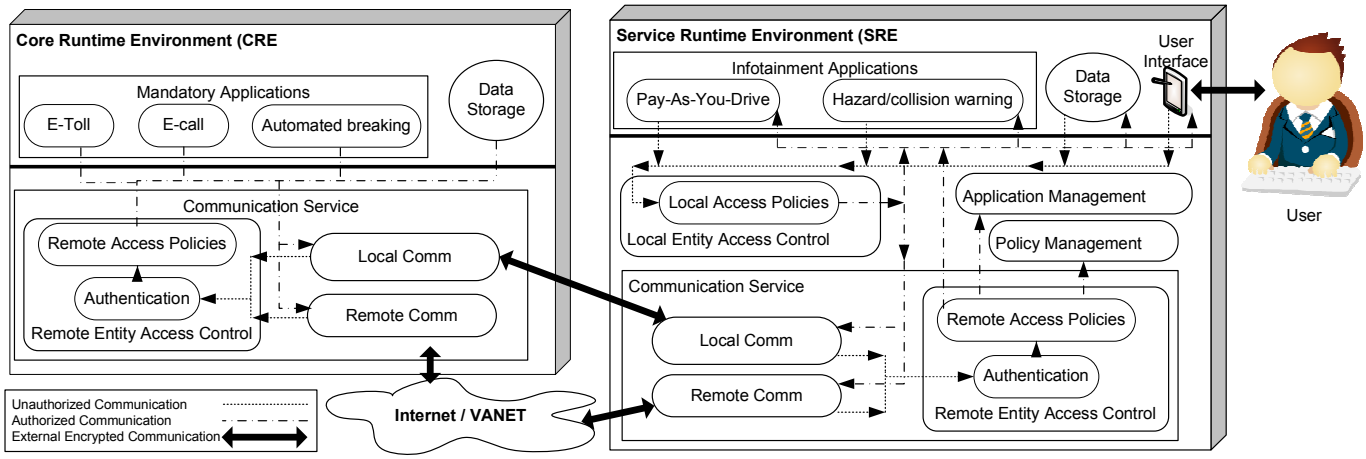

Fig. 2. Deployment view of the security architecture of the multi-application platform.

certain data collected in the CRE. This inter-runtime communication should be allowed according to predefined policies in the LEAC and REAC.

The Core Runtime Environment (CRE) hosts the critical and mandatory applications. It is a small, controlled and verified runtime platform in a tamper proof box. This runtime only hosts thoroughly verified and trusted applications which are pre-installed; the application composition is static but can be targeted to the type of vehicle. To reduce cost and increase integrity of this platform, the CRE will either not be updatable, or only updatable in a strictly controlled environment with the highest security standards, for example by a licensed car dealer.

The Service Runtime Environment (SRE) hosts the optional and non-critical applications such as infotainment applications. This environment can be tailored to the preferences of the user. The user can interact with the applications on the SRE through a user interface. This runtime hosts applications that can come from many different service providers and can therefore not be trusted. New applications can be installed on this platform at runtime. New applications are required to pass verification by the HMC before they can be deployed. The vehicle platform enforces service or user specific policies that cannot be enforced at verification time (e.g. inter application communication).

B. Local Entity Access Control (LEAC)

The LEAC is an interception engine that monitors all inter-application messages and applies access control on these messages. It contains a list of access policy items which state for each application which resources it is allowed to communicate with. The LEAC enforces these access policies. The policy engine can allow or deny a message from one application to another, and can log these messages in a database.

The LEAC facilitates *the integrity and confidentiality of data and applications*. By limiting the access to resources, applications can only gather and alter data and perform operations they are privileged to. Sensitive information and resources are only accessible to those applications that have received explicit permission.

The LEAC is only present on the SRE. The first reason is that possibly malicious or severely erroneous applications can only be installed on the SRE. The applications on the CRE are validated and completely trusted, so these do not have to be protected from one another. The second reason is that the LEAC causes extra overhead on the platform. For critical and mandatory applications this overhead needs to be avoided. The SRE on the other hand does not host critical applications so additional overhead is acceptable.

The LEAC is managed by the HMC. The HMC provides and updates the access policies. The user can be allowed to add additional policies as long as they do not conflict with policies provided by the HMC.

C. Remote Entity Access Control (REAC)

The REAC is responsible for authentication and authorization of incoming communication. Authentication can be done by having a list of certificates installed on the platforms. Every incoming message has to be signed to enable verification of the authenticity of the message. Once the identity of the sender has been checked, the REAC can activate access policies to allow or prevent message delivery. The REAC is also able to log the communication.

The REAC ensures the *integrity* of the applications and data by blocking incoming data from unauthorized parties. Especially the management applications have strict access limitations: only the HMC should be allowed to send new applications and new policies to the management applications.

On the CRE, a limited form of the REAC is needed. The applications are static and only a very limited amount of entities, which are known in advance, are allowed to send messages to the CRE. The REAC on the CRE does not have to change over time; all access policies are static.

On the SRE, the REAC has to be more elaborate. Due to the dynamic nature of the platform, the parties that are allowed to send messages to the platform change over time. The access

policies in the REAC also have to change over time. This configuration is done by the HMC.

V. Prototype Implementation

The proposed architecture is implemented on top of 2 run-time platforms. The Core Runtime Environment is implemented on an embedded Java CDC based platform. The Service Runtime Environment is implemented on a standard telematics platform, the CarPC, running an OSGi platform.

A. Core Runtime Environment (CRE)

The CRE is implemented on the NXP ATOP, an embedded Java ME CDC based platform. Java has been used for vehicle monitoring [7] and was chosen for its ease of programming, its inherent type security and industry wide support. It enables future evolution to other Java based platforms such as OSGi. It also allows for fast prototyping of the architecture.

The downside of this choice is the potential negative impact on performance and the larger binary size, which increases the cost of the platform. However the measured overhead remains limited and acceptable.

The current implementation is an event based component system. Each application is implemented in a component. These components can register themselves with an event manager. Components can send events to and receive events from other components and the SRE. A tolling application is created that gathers information from a GPS sensor and calculates the distance and the associated toll fee.

The Java ME CDC platform requires a minimal total storage memory of 192KB. The event based component system with communication to the SRE and the Internet requires another 43KB of memory. Additional applications require another few KB of memory depending on their functionality. For example the e-toll application requires another 3KB of memory. A platform with a few MB of memory is enough to support the CRE. Real-time support and updatability require some additional memory, but the overhead should be manageable.

B. Service Runtime Environment (SRE)

The SRE is implemented on a standard telematics platform running a modified version of the Felix OSGi platform [8].

OSGi is a modular multi-application environment. Each application is packed into a bundle that can separately be installed, started, stopped, updated and removed. Bundles can provide services by providing a service registry with a service interface and an object implementing the interface. Bundles can consume services of other bundles by requesting the service object for a service interface. Bundles can be updated while providing and consuming services of other bundles, which makes it a very modular and updatable environment.

OSGi is chosen as the platform for prototyping for three main reasons: (1) it provides a modular abstraction for applications, (2) it is an industry accepted standard, and (3) the Java environment allows for easy and fast development of a prototype. The Felix implementation was chosen because of its open source implementation which allows us to modify the platform. The downside to choosing OSGi however are the few inherent security flaws [5].

The runtime allows for remote installation and removal of bundles. A gateway to the CRE is implemented. The gateway translates the Java method calls to events and sends these events to the CRE. It also processes the events from the CRE and calls the necessary methods.

The basic Felix OSGi framework has a size of 625KB. This provides the basic OSGi functionality with a service registry, bundle repository and a basic shell. For use in a telematics system, an additional bundle was developed that provides a graphical user interface and the connection with the CRE. The overhead of this bundle is an additional 86KB. A tolling application has been made that shows the current toll on the user interface together with the current location and an option to instantly pay the current toll due. This application bundle has a size of 4KB.

A prototype implementation of the security features is added to the Felix OSGi framework. Bundles are now only permitted to import and export services if the bundle's manifest file clearly states that the bundle is allowed to import or export that service. In addition to the access check, the Local Entity Access Control is implemented by using the proxy design pattern. The object that a bundle receives when requesting a service is no longer the object that provides the service but rather a proxy. This proxy logs each call to the service and checks whether the call is allowed. An additional bundle is provided to change the access rights at runtime. These new features increase the size of the basic framework with 14KB.

Security issue	OSGi	GST	presented architecture
Availability of applications			*X
Confidentiality of applications	X	*	*X
Confidentiality of data	X	*X	*X
Integrity of applications	*X	*	*X
Integrity of data	X	*	*X

TABLE I
A comparison between a standard OSGi platform, the GST architecture and our proposed architecture.
*: protected from malicious outside parties
X: protected from malicious applications

VI. Related work

We position our research in the context of two service platforms that can be used for secure vehicle telematics platforms (GST and OSGi). First we introduce the two projects, then we compare the proposed architecture with a standard OSGi platform and the GST architecture (see Table I).

GST [3] is a European vehicle telematics research project. One of the sub-projects was concerned with securing telematics systems. They focus on protecting the vehicle platform against malicious outside parties while assuming that all applications executing on the vehicle can be trusted. However, when we consider the amount of telematics applications that can be installed, we can not make the assumption that these applications behave as expected when installed on a single

platform. To provide basic security, GST proposes to separate the execution environment in a standard runtime and a security module. The security module is responsible for secure data storage and cryptographic functions such as key agreement, secure logging and secure random number generation. The most security critical parts of some programs, such as online payment, also run in this security module. All other programs run in a standard runtime environment that interacts with the security module. However, all applications in the normal runtime can access the security module. This means that while the cryptographic keys themselves are not exposed, they can still be used without authorization.

OSGi is an industry standard for multi-application platforms that can be used as a telematics platform [5]. Although it has many advantages such as ease of use and providing inherent evolvability, key concerns can be identified when applied to vehicles. OSGi currently supports an adapted Java 2 Security functionality. Each bundle needs the appropriate Java Permission in order to access a service from the service registry or the file system. The security architecture is quite complex and often only partially implemented in the OSGi implementations. Researchers have investigated the current security of OSGi and concluded there are still significant weaknesses present on the platform [6]. Also, OSGi does not inherently offer any networking functionality.

Availability of applications: All OSGi applications typically run on a single platform, with no RAM or thread control. Any bundle can consume unlimited memory or spawn an infinite amount of threads, destroying the availability of the applications. GST does not mention any protection to guarantee availability for applications. In our solution, critical applications are executed in a separated and trusted environment. This prevents any malicious application from interrupting the operation of the platform or any critical application.

Confidentiality of applications: in OSGi, a bundle can only see those services it has permission to. GST does not mention any way to protect application confidentiality. It does protect resources from outside interference however. Our system protects application confidentiality by restricting access of resources to only those applications that have permissions.

Confidentiality of data: OSGi supports internal data confidentiality. Services are only accessible when having the required permission. GST protects critical data by storing it in a separated security module. Other data is only protected against outsider attacks. Our architecture ensures data confidentiality by restricting access. Only authorized applications or outside parties can query confidential resources and only they have access to the confidential data.

Integrity of applications: OSGi protects the integrity of the platform by a system of bundle signing, where only certified bundles can be installed on to the OSGi platform. GST verifies authenticity of incoming data so an unauthorized outsider cannot change the platform. Our platform restricts the usage of the management applications to only those parties that are authorized to use the management applications. Only the HMC is allowed to interact with the management applications.

Integrity of data: OSGi provides integrity of data from internal attack by restricting the use of services with Java Permissions. GST protects the data from outsider attack by authorizing incoming requests. We protect the most critical data by storing it in on the secured platform, and protect other data by authorizing any request on a resource.

VII. Conclusion

This paper presented a dual runtime environment architecture to enhance the confidentiality, integrity and availability of critical and mandatory applications while still providing an open platform for infotainment applications. The Core Runtime Environment (CRE) hosts the mandatory applications separated from the Service Runtime Environment (SRE) which can host an endless variety of infotainment applications. A connection between the two runtimes allows applications on the SRE to make use of the data on the CRE. The SRE is updatable to allow the platform to evolve in order to meet future changes. Additionally a Local Entity Access Control and a Remote Entity Access Control have been introduced to protect the confidentiality and integrity of the applications and data on the platforms. A proof of concept was implemented that demonstrates the feasibility of the architecture.

The architecture only offers a mechanism to enforce policies. However, a point of interest is how to create and manage a consistent and non-conflicting set of policies based on government, HMC and Service Center requirements and the wishes of the multiple users of the car.

At the moment we only assume communication over the Internet. Yet vehicles will also communicate with other vehicles and road-side infrastructure. Another point of interest is how we can securely provide and consume services to the other vehicles or the infrastructure.

Acknowledgements

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT and the Research Fund K.U. Leuven. We also would like to thank NXP for providing the ATOP hardware.

References

[1] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *Computer*, vol. 35, no. 1, pp. 88–93, Jan 2002.

[2] R. Hsu and L. Chen, "An integrated embedded system architecture for in-vehicle telematics and infotainment system," in *Industrial Electronics, 2005. ISIE 2005. Proceedings of the IEEE International Symposium on*, vol. 4, 20-23, 2005, pp. 1409–1414.

[3] "Gst home." [Online]. Available: http://www.gstforum.org/

[4] L. Feast, J. Yoon, and R. Winchester, "Toyota braces for sales hit from recall," 2010. [Online]. Available: `http://www.reuters.com/article/idUSTRE6100KS20100202`

[5] D. Zhang, X. H. Wang, and K. Hackbarth, "Osgi based service infrastructure for context aware automotive telematics," in *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, vol. 5, May 2004, pp. 2957–2961 Vol.5.

[6] P. Parrend and S. Frenot, "Security benchmarks of osgi platforms: toward hardened osgi," *Softw. Pract. Exper.*, vol. 39, no. 5, pp. 471–499, 2009.

[7] J. Hua, L. Xu, X. Lin, M. Hu, J. Li, and M. Ouyang, "Mems and j2me based acceleration real-time measurement and monitoring system for fuel cell city bus," in *Mechatronics, 2009. ICM 2009. IEEE International Conference on*, April 2009, pp. 1–6.

[8] "Felix osgi." [Online]. Available: http://felix.apache.org/